**淘攻略Push**

* 业务介绍：

Push业务是通过分析内容搜忠实用户的行为，找到全网相似的用户，推送他们可能感兴趣的文案。

文案是以模板形式给出，需要填充一个特定的类目词。算法的任务是计算出在什么时候推送给什么人什么类目词。

* 对接人：手淘push运营：伊聆；push全局调度：米砾。钉钉群：淘攻略push合作。
* 整体流程：push数据是离线产出，部署在odps的任务节点。
* 数据格式：最后产出的数据结果，

User\_id:用户id，os：操作系统，feed\_id:为了和头条push兼容，我们的是空，item\_attr:所有乱七八糟的属性，在生成文案时要用的。Event:和头条兼容，我们的是空，score，personal score：排序的分数，用在截取前500万个和push投放时全局优化用。

* 技术细节：push的大概流程如下



圈人：圈人是我这边算法主要负责，计算什么人在推什么类目，以天级别，主要分析用户的浏览商品行为，结合用户画像和类目画像，以及该类目的内容性，计算是否推荐用户这个类目。



计算用户对类目的内容偏好：

使用一个分类模型。这里计算对商品叶子类目的偏好。正样本是淘攻略忠实用户在搜索该类目商品后有阅读淘攻略文章的行为；负样本是搜索商品后直接购买，没有阅读文章。

特征抽取：

用户的基础特征，是从主搜的特征表中得到，相对单一。大概只有性别、年龄、地域、购买力等基本特征。在主搜特征表中通过类目为0过滤得到。

类目特征，首先通过用户购买生成类目协同矩阵。对这个矩阵降维后作为类目特征。

实时特征，用户最近一段时间的商品浏览行为，我们使用了小时表做为这部分数据的来源。15分钟表存在不稳定的问题，回流不稳定。Push算法的效果很大部分取决于数据的时效性。所以兜底方案是使用前一天的数据，而非兜底方案使用节点运行前当天的数据。用来提高这部分特征的时效性，以提高push打开率。

* 效果评测：通过线上的push打开率比历史提升来看。上新的算法需要灰度测试。
* 代码：dii代码在alimama\_sv\_lab项目空间hongshu文件夹：

predict\_bianry：spark任务+PS任务，xgboost预测模型。

merge\_to\_product\_tb：产出最终的表，做了文档召回、违禁类目、人工黑名单过滤，并限定了分数top 500万作为产出。

对应的兜底节点逻辑一样，只是前一天晚上产出。

Push\_doudi文件夹：在晚上11点跑的，为第二天做兜底。

predict\_bianry\_doudi：spark任务+PS任务，xgboost预测模型。

merge\_to\_product\_tb\_doudi：产出最终的表，做了文档召回、违禁类目、人工黑名单过滤，并限定了分数top 500万作为产出。

merge\_push\_all\_doudi：循环检测当天任务是否顺利完成，如果在8点半前没有完成则使用前一天的兜底数据。

* 代码：
* 报表：

线上投放效果：淘攻略srp：

运行d2任务，在云端路径：search\_kg\_dev空间/鸿数/push\_di/push报表。把其中两个select的语句的输出结果贴在excel里整理。

**淘攻略 query suggestion**

* 业务介绍：query suggest是在淘攻略的query输入框中的下拉提示。和主搜的query下拉最大的区别是，淘攻略的query下拉需要更多的内容型query，而主搜原来的下拉都是商品型query。
* 对接人：目前这块东西很稳定。在线上复用主搜query下拉的排序模块，我负责生成query和给query打分的离线表。线上d2模块和 洛离 对接，算法有问题请教开锋。
* 整体流程：

生成一张query-score的表，上到主搜query下拉引擎的d2上。复用一下主搜稳定的query下拉服务链。Bts需要在service condition中配置。

* 技术细节：

生成内容query：



QT对很容易得到，query 点了title，就生成QT对。QQ对是用户rewrite了query生成的。通过分为增删生成term->term的关系。删除是到空节点。关系构成图。对图做pagerank。最后结合top query按类目做扩展，生成内容query。并按照一个概率转移公式计算query的分数。

* 包含节点：search\_kg\_dev/鸿数/query\_suggestion/query\_suggestion\_content\_query\_extention
* 主要实现论文；Query Suggestion by Constructing Term-Transition Graphs.pdf

**淘攻略主搜query透出**

* 计算逻辑：根据用户log分析不同query展示什么样的文档类型，存入表kg\_cbir\_trigger\_query\_all。@葭禾 会取表中的数据作为主搜query透出淘攻略内容的trigger。
* 节点：query\_sel\_for\_main\_search
* 对接人：葭禾
* 代码：/jimmy.zjm/query-suggestion
* 类：QuerySelForMainSearch

**内容结构化**

* **类目搭建**
* 概述：内容类目是从淘宝头条团队的运营类目衍生出来的，通过多名运营的分工整理。算法通过全淘宝内容库召回文档计算标签质量。反馈运营后，运营增删类目，调整类目结构。
* 对接人：不欺
* 算法模块：计算所有类目词在2000万篇文章中的召回量，和父节点的卡方相关度。在人工调优权重，算出最后的质量分。分到4个档位：推荐，一般，不推荐，极差。（spark）。之后人工会根据算法的打分判断是否要保留这个类目。
* 算法代码：/jimmy.zjm/query\_process

spark类：com.taobao.search\_kg.content\_tagging.DocLabelScore，run-label-score.sh

* 结果：/jimmy.zjm/content-tagging

Categories\_input: 运营整理的类目

Algorithm\_score: 算法打分

Categories\_modify: 最后运营根据算法打分修改的类目

* 更多背景知识：/jimmy.zjm/content-tagging/wikis/home
* **属性整理**
* 技术方案：整个内容结构化的方案参考附件的ppt。属性目前是自由散乱的，但要归属到顶级类目。准确说目前只是挖掘了很多属性值，并没有规约到属性。

属性值的挖掘也是机器和人工交互进行。首先，在外网（主要是今日头条）挖掘别人整理好的标签，挖掘的方式是“顶层类目词 属性值词”传入今日头条搜索引擎，取召回的文章结果。召回的文章一般在末尾会有几个tag，提取这些tag，聚合后看是否和顶层类目相关性较强，召回文章是否足量等，再做敏感词过滤。剩余的tag提供给运营再次过滤，然后上线。属性值词的启动词是运营整理的，每个类目200个左右。

* 爬取tag代码：11.180.154.36 /home/jimmy.zjm/workspace/content\_search/scripts/crawler/toutiao\_tags\_crawler.py
* 启动词：/content-tagging.git D:\workspace\content-tagging\其他\种子词全类目0515.xlsx

数码的单独放在D:\workspace\content-tagging\其他\数码属性词.tags

* 敏感词过滤是夫卡提供的离线违禁词表，在git项目中。
* 目前还存在类目重刷的问题，因为整个任务目前是离线15分钟增量完成，头条后台索引建在内存，所以没有办法重新刷新tag。但好在展示的文章都比较新，旧的文章很快就被刷掉，影响不是很大。

目前 元坎 提供了一个方案，每天我这边找出label有变动的文章，他那边重刷。具体计算逻辑：在云端节点toutiao\_label\_modify\_track。

具体细节参考ppt。

* **文章分类&文章打标**
* 介绍：文章分类达标目前落地场景是头条标签搜索。整个链路：我负责给文章达标（odps离线任务），
* 对接人：上游及汇入数据：成平；后台增量：元坎；前端：艺璇；运营：不欺，若妍；pd：tianlan；搜索srp：招提
* 算法流程：文章分类 + 打对应类目的标签，标签式通过上面的属性整理产出的。
* 离线打标任务：因为头条文章在最后展示之前需要过审核，15分钟会生成一批新的文章。在过审核之前我的离线节点给文章打标。因为更新jar包如果发生异常影响较大，所以才用主从节点的模式。主节点在alimama\_sv\_lab下的hongshu /toutiao,从节点在hongshu/toutiao/score\_label。一般线上如果跑的是主节点的udf，则先上线从节点udf，验证稳定后，在toutiao\_content\_tagging\_v1 中切换上线。

Toutiao\_label\_modify\_track是负责统计每天新增的和发生tag改变的文章，提供给元坎重刷数据。

Udf所需要的数据在下面预测代码的git中，并没有都加入到在云端。

* Top类目分类：top类目通过fasttext分类器分类。
* 调用方式：predict\_content\_category(String content, Long max\_tags, Boolean display\_prob)

max\_tags：保留前几个tag。当传入为0时，输出预测类目结果。

display\_prob：（true, false）当max\_tags不为0时生效，是否输出tag的prob。

* 预测代码： [/jimmy.zjm/fasttext-cbir-feed-tagging](http://gitlab.alibaba-inc.com/jimmy.zjm/fasttext-cbir-feed-tagging)
* 训练代码：jimmy.zjm/fasttext-cbir-feed-tagging 中的train，test文件。使用fasttext unigram训练。
* 技术细节：通过前面的采集tag的方式，可以得到tag。新建了一个odps的udf函数predict\_content\_category，当文档传入后，先用fasttext预测top类目，再把最大可能的类目所有tag和文章匹配，因为直接查找比较慢，如果以后词表大了会有性能问题，如果之后要做成线上模块也有问题。所以只把uni和bi-gram的标签做匹配，这样只需要用set存好就可以了，满足了99%的标签匹配。匹配好的标签通过tf-idf打分，排序，过滤敏感词，返回结果。